MapReduce


Introduction

How to write software for a cluster?
- 1000, 10,000, maybe more machines
  - Failure or crash is not exception, but common phenomenon
- Parallelize computation
- Distribute data
- Balance load
- Makes implementation of conceptually straightforward computations challenging
  - Create inverted indices
  - Representations of the graph structure of Web documents
  - Number of pages crawled per host
  - Most frequent queries in a given day

MapReduce

Abstraction to express computation while hiding messy details
- Inspired by map and reduce primitives in Lisp
  - Apply map to each input record to create set of intermediate key-value pairs
  - Apply reduce to all values that share the same key (like GROUP BY)
- Automatically parallelized
- Re-execution as primary mechanism for fault tolerance

Programming Model

- Transforms set of input key-value pairs to set of output key-value pairs
- Map written by user
  - Map: (k1, v1) → list (k2, v2)
- MapReduce library groups all intermediate pairs with same key together
- Reduce written by user
  - Reduce: (k2, list (v2)) → list (v2)
  - Usually zero or one output value per group
  - Intermediate values supplied via iterator (to handle lists that do not fit in memory)

Example

Count number of occurrences of each word in a document collection:

```
map( String key, String value ):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate( w, "1" );

reduce( String key, Iterator values ):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt( v );
  Emit( AsString( result ) );
```

This is almost all the coding needed...
(need also mapreduce specification object with names of input and output files, and optional tuning parameters)
Implementation

- Focuses on large clusters
  - Relies on existence of reliable and highly available distributed file system
- Map invocations
  - Automatically partition input data into $M$ chunks (16-64 MB typically)
  - Chunks processed in parallel
- Reduce invocations
  - Partition intermediate key space into $R$ pieces, e.g., using $\text{hash(key)} \mod R$
- Master node controls program execution

Fault Tolerance

- Master monitors tasks on mappers and reducers: idle, in-progress, completed
- Worker failure (common)
  - Master pings workers periodically
  - No response $\Rightarrow$ assumes worker failed
    - Completed tasks only on local disk, hence inaccessible
  - Same for reducer’s in-progress tasks
  - Reduced tasks stored in global file system, hence accessible
  - Reducers notified about change of mapper assignment
- Master failure (unlikely)
  - Checkpointing or simply abort computation

Applicability of MapReduce

- Machine learning algorithms, clustering
- Data extraction for reports of popular queries
- Extraction of page properties, e.g., geographical location
- Graph computations
- Google indexing system
  - Sequence of 5-10 MapReduce operations
  - Smaller simpler code (3800 LOC -> 700 LOC)
  - Easier to change code
  - Easier to operate, because MapReduce library takes care of failures
  - Easy to improve performance by adding more machines

MapReduce vs. DBMS

- Map: assume table “InputFile” with schema (key1, val1) is input; “mapFct” is a user-defined function that can output a set with schema (key2, val2)
  ```sql
  SELECT mapFct( key1, val1) AS (key2, val2) // Not really correct SQL
  FROM InputFile
  ```
- Reduce: assume MapOutput has schema (key2, val2); redFct is a user-defined function
  ```sql
  SELECT redFct( val2 )
  FROM MapOutput
  GROUP BY key2
  ```
**Parallel DBMS**

- SQL specifies what to compute, not how to do it
  - Perfect for parallel and distributed implementation
  - “Just” need an optimizer that can choose best plan in given parallel/distributed system
    - Cost estimate includes disk, CPU, and network cost
- Recent benchmarks show parallel DBMS can significantly outperform MapReduce
- But many programmers prefer writing Map and Reduce in familiar PL (C++, Java)
- Recent trend: High-level PL for writing MapReduce programs with DBMS-inspired operators

**MapReduce Summary**

- MapReduce = programming model that hides details of parallelization, fault tolerance, locality optimization, and load balancing
- Simple model, but fits many common problems
- Implementation on cluster scales to 1000s of machines and more
- Open source implementation, Hadoop, is available
- Parallel DBMS, SQL are more powerful than MapReduce and similarly allow automatic parallelization of “sequential code”
  - Never really achieved mainstream acceptance or broad open-source support like Hadoop
- Recent trend: simplify coding in MapReduce by using DBMS ideas
  - (Variants of) relational operators, implemented on top of Hadoop

**SQL Injection**

- Exploits security vulnerability in database layer of a Web application when user input is not sufficiently checked and sanitized
  - Think DBMS access through Web forms
- Main idea: pass carefully crafted string as parameter value for an SQL query
  - String executes harmful code
    - Reveals data to unauthorized user
    - Data modification by unauthorized user
    - Deletes entire table
- The following examples are from unixwiz.net

**Getting Started**

- Assume we know nothing about Web application, except that it probably checks user email with query like this:

  ```sql
  SELECT attributeList
  FROM table
  WHERE attribute = '$email';
  ```

- Typical for Web form allowing user login and send password to user’s email address
  - $email is email address submitted by user through Web form
  - Try entering `name@xyz.com` in form:

  ```sql
  SELECT attributeList
  FROM table
  WHERE attribute = 'name@xyz.com';
  ```

**First Code Injection**

- Query has incorrect SQL syntax
  - Getting syntax error message indicates that input is sent to server unsanitized
- Now try injecting additional “code”:

  ```sql
  SELECT attributeList
  FROM table
  WHERE attribute = 'x' AND email IS NULL;
  ```

- Legal query whose WHERE clause is always satisfied
- Might see response from system like “Your login info has been sent to somebody@somewhere.com”
- Enough information to start exploring the actual query structure

**Guess Names of Attributes**

- Try if “email” is the right attribute name:

  ```sql
  SELECT attributeList
  FROM table
  WHERE attribute = 'x' AND email IS NULL;
  ```

- Server error would indicate that attribute name “email” is probably wrong; if so, try others
- Valid response (e.g., “Address unknown”) indicates that attribute name was correctly guessed
- Can guess names of other attributes like “passwd”, “login_id”, “full_name” and so on
**Guess Table Name**

- Try if “tabname” is a valid table name:

  ```sql
  SELECT attributeList FROM table WHERE attribute = 'x' AND 1 = (SELECT COUNT(*) FROM tabname); --;
  ```

- If no server error, found valid table name, e.g., “members”
- But is it the name of the table used for the query behind the Web form?

**Find Table Name for Unknown Query**

- Try query that only works if table “members” is part of the query:

  ```sql
  SELECT attributeList FROM table WHERE attribute = 'x' AND members.email IS NULL; --;
  ```

- Error like “Email address unknown” indicates that query was syntactically correct, i.e., “members” is a table in the FROM clause

**Finding Users**

- Look on application’s Web pages to find names of people, then find them in the database (recall that full_name was found to be an attribute):

  ```sql
  SELECT attributeList FROM table WHERE attribute = 'x' OR full_name LIKE '%Bob%';
  ```

- If server returns message like “Sent your password to bob@example.com”, found some Bob’s email in database

**Guessing Passwords**

- Try password through same query form (recall that passwd was found to be an attribute):

  ```sql
  SELECT attributeList FROM table WHERE attribute = 'bob@example.com' AND passwd = 'pwd123';
  ```

- Found password when “Your password has been mailed to …” message appears
- Tedious guessing procedure, but can be automated with script

**Deleting a Table**

- Inject a DROP TABLE statement for the table names found earlier:

  ```sql
  SELECT attributeList FROM table WHERE attribute = 'x'; DROP TABLE members; --;
  ```

- …and table “members” is gone, unless permissions do not allow it to be deleted by Web app.

**Adding a New Member**

- Inject an INSERT statement like the DROP TABLE statement before
- Possible problems:
  - Input string length in Web form might be limited
  - Web app might not have insert permission
  - Some attribute names might be unknown still, and might require values in the INSERT
  - Foreign key relationships, CHECKs etc might require other updates before new member tuple can be inserted
- So, let’s try something different...
Modify Existing Tuples

- Replace email address to get password mailed to new address:

  ```sql
  SELECT attributeList
  FROM table
  WHERE attribute = 'x';
  UPDATE members
  SET email = 'myEmailAddress'
  WHERE email = 'bob@example.com';
  ```

- Then use the "Email me my password" link
  - Now have access to the system as Bob, who probably is important (if his name was mentioned as Web admin etc.)

Preventing SQL Injections

- Sanitize form input received from users
  - Only allow characters that could occur in email address (or whatever the form field is for)

- Escape/quotesafe the input (prevent illegal use of ' character)
  - Name like O'Reilly is legal string 'O''Reilly', but "WHERE name = '\"'; DROP TABLE members;--'" should be prevented
  - Difficult, but functions exist for identifying if something is an escape string

Preventing SQL Injections

- Use bound parameters (PreparedStatement)

  ```java
  PreparedStatement ps = con.prepareStatement("SELECT email FROM member WHERE name = ?");
  ps.setString(1, formField);
  ResultSet rs = ps.executeQuery();
  ```

  - Any code injected into form field will just be part of the name field's value
  - Works similarly if email is input field of stored procedure

Preventing SQL Injections

- Limit database permissions for Web app

- Isolate the Web server
  - Even if Web server is compromised by SQL injection, make sure it cannot do much harm

- Properly configure error reporting
  - Do not output developer debugging information on unexpected inputs

Final Comment

From xkcd.com/327

From xkcd.com/327